Secure Ownership and Ownership Transfer in RFID Systems

Ton van Deursen[1⋆], Sjouke Mauw[1], Saša Radomirović[1], and Pim Vullers[1,2]

[1] University of Luxembourg, Luxembourg.
{ton.vandeursen, sjouke.mauw, sasa.radomirovic}@uni.lu
[2] Radboud University Nijmegen, The Netherlands.
p.vullers@cs.ru.nl

Abstract. We present a formal model for stateful security protocols. This model is used to define ownership and ownership transfer as concepts as well as security properties. These definitions are based on an intuitive notion of ownership related to physical ownership. They are aimed at RFID systems, but should be applicable to any scenario sharing the same intuition of ownership.
We discuss the connection between ownership and the notion of desynchronization resistance and give the first formal definition of the latter. We apply our definitions to existing RFID protocols, exhibiting attacks on desynchronization resistance, secure ownership, and secure ownership transfer.

Key words: RFID protocols, ownership, desynchronization resistance, ownership transfer, formal verification

1 Introduction

Radio frequency identification (RFID) is expected to become a key technology in supply chain management, because it has a large potential to save costs. Two of the cost-saving advantages of this technology are the improved efficiency of inventory tracking and the reduction of counterfeit products. The former is due to the fact that RFID is contactless and requires no line of sight between the RFID reader and the RFID tag attached to a product. The latter is because RFID tags can store and process information as well as execute simple communication protocols.

As products flow through a supply chain, their ownership is transferred from one partner to the next. This transfer of ownership extends to the RFID tags attached to these products. This means that at some point in time a supply chain partner owns the products and RFID tags legally, by means of a title, and physically by the fact that the goods are at his premises. In general, ownership of an object allows one to (exclusively) interact with the object, modify the object, and transfer ownership of the object to someone else.

⋆ Ton van Deursen was supported by a grant from the Fonds National de la Recherche (Luxembourg)

In this work, we propose and attempt to validate a definition of ownership in RFID systems, which is inspired by the legal and physical meaning of ownership. We use this definition as a basis to define secure ownership, in Section 3, and secure ownership transfer in RFID protocols in Section 4. These definitions are particularly relevant for RFID systems in supply chains, but we expect them to be also applicable to other scenarios that share the same intuition of ownership, such as future parcel delivery systems. The definitions of these properties are, to the best of our knowledge, the first formal definitions proposed. We attempt to validate them by considering a published protocol designed for ownership transfer. We exhibit a flaw in the protocol and demonstrate attacks on secure ownership and secure ownership transfer.

2 Stateful Security Protocols

In this section we introduce basic notation and definitions concerning security protocols. Rather than providing a full description of security protocol syntax and semantics, we only present the essentials needed for defining and analyzing ownership and related notions. A more extensive description can be found in Appendix A. The model presented is based on the model for stateless protocols by Cremers and Mauw [1]. We extend their model by adding support for stateful protocols. While stateless protocols start in the same state for every execution, stateful protocols may use information from previous and parallel protocol executions.

A *protocol* is defined as a map from an n-tuple of distinct *roles* to an n-tuple of *role specifications*. A role specification defines the behavior of an *honest agent* executing the role. Typical roles in an RFID system are the reader and tag roles to be executed by actual RFID readers and RFID tags. A particular execution of a protocol role by an agent is called a *run*.

The specification consists of a composition of events and the declaration of all nonces and variables appearing in the composition. An *event* is either the sending or the receiving of a message and both can be accompanied by assignments to variables. The receiving of messages is referred to as a *read* event. Inspired by Ryan et al. [2], we use *signals* to indicate that a certain point in the protocol has been reached.

The exchanged messages between roles consist of *terms*. These terms are built from basic terms such as nonces, constants, and agent names. Complex terms can be constructed using functions like $\{\cdot\}_\cdot$ (encryption), $h(\cdot)$ (hashing), $\cdot \oplus \cdot$ (exclusive or), and $(\cdot, \cdot)$ (pairing). When an agent executes a role, nonces are freshly generated and variables receive their actual value through read events and assignments. We separate two kinds of variables. *Local variables* model the stateless part of protocols. Their values are assigned through read events and they are reassigned every run. Once assigned, their value does not change. The stateful part of protocols is modeled by *global variables*. They receive their value through explicit assignments and their values are maintained across different runs.

We study the possible behavior of a system in which a collection of agents executes a set of protocols Π through so-called *traces*, denoted by $\mathsf{traces}(\Pi)$. Informally, a trace is a list of events occurring in the interleaved execution of protocol runs. The precise construction of traces is dictated by the semantics of the system (given in Appendix A). Formally, a trace $t = t_0 \dots t_{n-1}$ is a valid derivation $s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} \dots \xrightarrow{t_{n-1}} s_n$ of system states $s_0 \dots s_n$ and events $t_0 \dots t_{n-1}$, and $|t| = n$ is its length. Abusing notation, we write $\Sigma(t)$ to denote the states $s_0 \dots s_n$ of trace t.

A system state is a five-tuple It contains the following components. The set A is used to record active runs. Each run contains an identifier, the name of the executing agent, the list of events that still have to be executed, and the local variable assignments. A run r has been completed successfully in state s, denoted by $\mathsf{success}(r, s)$, if its event list is empty. Otherwise the run is still active or it has terminated unsuccessfully.

The current state of the global variable assignments of the agents is stored in G. We consider communication to be asynchronous. Messages sent by agents are placed in the send buffer SB. Similarly, agents read message from the read buffer RB. Finally, the intruder's knowledge is kept in I.

We assume that a standard Dolev-Yao intruder [3] controls the network. The intruder delivers a message by moving it from the send buffer to the read buffer. He eavesdrops on messages by adding them to his knowledge. The intruder can construct any message from his knowledge and place it in the read buffer. He can block or delay messages by not moving them from the send to the read buffer. Finally, a message can be modified by faking a message and blocking the original one. As usual in Dolev-Yao intruder models, we assume that cryptography is perfect. This means that the intruder cannot reverse hash functions and that he is not able to learn the contents of an encrypted term, unless he knows the decryption key. We assume that there is one agent E which is under full control of the intruder.

We use message sequence charts [4] to represent protocol specifications graphically. Every message sequence chart shows the role names, framed, near the top of the chart. Above a role name, the role's secret terms are shown. Actions, such as nonce generation, computation, verification of terms, and assignments are shown in boxes. Messages to be sent and expected to be received are specified above arrows connecting the roles. It is assumed that an agent continues the execution of its run only if it receives a message conforming to the specification.

3 Ownership

In this section we consider two views on tag ownership. The first view, which we call the system view, is, that ownership of a tag is the ability to interact with the tag in a predefined manner. Ownership of a tag can, for instance, be defined as an agent's ability to inspect the tag's ID. The second view is called the agent view. It is based on the fact that each agent records in a local data structure the

tags it believes to be the owner of. We state a relation between these two views as a security requirement.

3.1 System View of Ownership

We define ownership of a tag as the ability to execute a designated protocol with the tag. This could, for example, be a mutual authentication protocol or a tag identification protocol. We call this protocol the *(ownership) test protocol*. This approach has been chosen over a knowledge-based solution, in which knowledge of a secret on the the tag indicates ownership, because it is more general. It allows, for example, to include trusted or other third parties in the decision of ownership.

We note that the test protocol does not have to be implemented on the tag. It is merely used to *define* what constitutes an owner of a tag and may thus be a virtual protocol. Consequently, in every state of the system, the ownership relation between tags and other agents is precisely defined, while the (hypothetical) executions of the ownership test protocol are not part of the system's traces. Ownership is tested in a virtual environment, consisting only of the testing agent, tag, and other agents specified by the test protocol's roles, but without any adversarial influence. The ability of the testing agent to successfully complete the test protocol proves ownership of a tag. In some contexts the knowledge of a key may be the defining notion of ownership, while in others it may be the ability to execute some or all protocols implemented on a tag. In the former setting, a simple proof-of-knowledge protocol would be a suitable test protocol, in the latter setting it would be the collection of protocols implemented on the tag.

A consequence of our approach to define ownership relative to a test protocol is that notions such as *ownership transfer* are also relative to the chosen test protocol. The choice of a proper test protocol is therefore an important step in all verification efforts. Choosing an insufficient test protocol may lead to ownership-related vulnerabilities being overlooked. A trivial example is the test protocol that can be successfully executed by any agent and which thus declares everyone as the owner of a tag. This problem is, however, mitigated by the fact that an intuitive notion of ownership frequently coincides with the ability to complete a mutual authentication protocol with a tag. In such cases, the authentication protocol can simply be taken to be the test protocol.

Testing for ownership of a tag in state s amounts to verifying whether the test protocol can be executed in a virtual environment whose initial state is s. In order to model this, we introduce the notion of *micro traces*. These can be derived from the traces described in Section 2 by allowing only one run for each of the parties involved and disallowing intruder activities.

We denote by $\mu\mathsf{traces}_{P(a_1,\ldots,a_n)}(s)$ the micro traces for protocol P when executed by agents $a_1 \ldots a_n$, starting from initial state s. For every role, we allow the creation of exactly one run. Since we do not verify security claims in micro traces, but rather define ownership, no intruder is modeled. Therefore all messages sent from one agent to another are delivered.

We now have all ingredients to formally define ownership.

Definition 1 (Tag Owner). *Let $\mathcal{A}$ be a projection from system states to active runs. An agent R is owner of tag T with respect to test protocol P in system state s, denoted by* $\mathsf{owns}_P(R, T, s)$*, if and only if*

$$\exists_{t \in \mu\mathsf{traces}_{P(R,T)}(s)} \forall_{r \in \mathcal{A}(\Sigma(t)_{|t|})} \mathsf{success}(r, \Sigma(t)_{|t|}).$$

Informally, an agent R *owns* a tag T *with respect to a test protocol* P, if in absence of all adversarial activity, R and T can successfully complete the protocol P. In this context, R is called the *owner* of T with respect to P and T is called R's *property* with respect to P.

We stress that our definition of ownership is not the definition of a security requirement. Our notion of ownership is merely used as a basis to define certain security requirements, in particular secure ownership and secure ownership transfer.

3.2 Agent View of Ownership

The definition of tag ownership allows one to verify whether an agent owns a tag. It misses, however, the owner's point of view. This view is important when discussing the intention of an owner to transfer ownership, i.e. the fact that the owner engages in an ownership transfer protocol. Thus we introduce the agent's view regarding ownership of a tag by defining tag *holders*.

A tag holder is an agent which, based on its protocol executions and local data structure, believes it is the owner of a tag. We model whether an agent holds a tag T with respect to test protocol P by a variable $holds(P, T)$.

Definition 2 (Tag Holder). *Let s be a system state $\langle A, G, SB, RB, I\rangle$ such that $G(R) = \sigma$ for an agent R. We call R a holder of tag T with respect to test protocol P in system state s, denoted by* $\mathsf{holds}_P(R, T, s)$*, if and only if*

$$\sigma(holds(P, T)) = true.$$

By modeling tag holding explicitly we can let the protocol execution depend on the value of the *holds* variable. This allows us, for instance, to specify that an agent shall not transfer ownership of a tag, unless it actually holds the tag.

For verification purposes, we decorate protocols in which a role changes the value of the *holds* variable with two signals: *obtain* and *release*. The obtain signal indicates an assignment of *true* to the *holds* variable, while the release signal indicates an assignment of *false*. We discuss these signals in more detail in Section 4.1.

3.3 Secure and Exclusive Ownership

In an ideal world, the notions of tag owner and tag holder coincide. It is, however, immediate that this is impossible to achieve in an asynchronous communication model. Tag ownership changes when a tag updates its knowledge. Due

to asynchronicity, an agent is in general not be able to update its *holds* variable simultaneously with the ownership change.

We define *secure ownership* as a consistency requirement on all states. We say that a set of protocols provides secure ownership, if, whenever an agent is holder of a tag, it must also be the owner of that tag.

Definition 3 (Secure Ownership). *A set of protocols Π provides secure ownership with respect to test protocol P if and only if*

$$\forall_{t \in \mathsf{traces}(\Pi)}\ \forall_{0 \le i \le |t|}\ \forall_{R,T \in \mathsf{Agent}}\ \mathsf{holds}_P(R, T, \Sigma(t)_i) \Rightarrow \mathsf{owns}_P(R, T, \Sigma(t)_i).$$

Secure ownership provides a guarantee to the owner that it cannot be "disowned" as long as it holds a tag. But secure ownership does not guarantee that no other agent can have simultaneous ownership of the tag. Simultaneous ownership is prevented by the notion of *exclusive ownership*. It guarantees that the holder of a tag is the sole owner of the tag. This is important, for instance, when nobody (and in particular no previous owner) but the holder of a tag is supposed to be able to identify or trace a tag. We define *exclusive ownership* as the requirement that if an agent holds a tag, no other agent is owner of the tag.

Definition 4 (Exclusive Ownership). *A set of protocols Π provides exclusive ownership with respect to test protocol P if and only if*

$$\forall_{t \in \mathsf{traces}(\Pi)}\ \forall_{0 \le i \le |t|}\ \forall_{R,T \in \mathsf{Agent}} \mathsf{holds}_P(R, T, \Sigma(t)_i) \Rightarrow \neg \exists_{R' \in \mathsf{Agent} \setminus \{R\}}\ \mathsf{owns}_P(R', T, \Sigma(t)_i).$$

It is clear that in an environment where owners can trace tags, exclusive ownership is a necessary condition for ownership transfer protocols to satisfy untraceability against previous and future owners of tags.

4 Ownership Transfer

In this section we define the notion of an ownership transfer protocol and the natural security requirement for such a protocol. We call a protocol Q an *ownership transfer protocol* if it satisfies the following functional requirement. By executing Q an agent can become the owner of a tag, if it has not been the owner of the tag.

Definition 5 (Ownership Transfer Protocol). *Let P be an ownership test protocol. We say that $Q \in \Pi$ is an ownership transfer protocol with respect to P if and only if*

$$\exists_{t \in \mathsf{traces}(\Pi)}\ \exists_{0 \le i < |t|}\ \exists_{R,T \in \mathsf{Agent}}\ \neg \mathsf{owns}_P(R, T, \Sigma(t)_i) \wedge \mathsf{owns}_{Q \cdot P}(R, T, \Sigma(t)_i),$$

where $Q \cdot P$ is used to denote sequential protocol composition.

Informally, the definition states that Q is an ownership transfer protocol, if there exists an agent R for whom the following two conditions are met. First, R is not an owner of T and hence cannot successfully complete the protocol P with T. Second, R is able to successfully complete the sequential composition of Q followed by P with a tag T.

4.1 Signals

In order to reason about the agent's view of ownership in a transfer protocol, we need to keep track of the events in a trace in which an agent changes the value of the *holds* variable. For this purpose we decorate protocols with *obtain* and *release* signals as follows. We identify the assignment of *true* to the *holds* variable with the appearance of an *obtain* signal and the assignment of *false* with the appearance of a *release* signal. For a trace $t = t_0 \dots t_{n-1}$, $0 \leq i < n$, we write $t_i = \mathsf{obtain}_P(B, T, A)$ to denote any event of a run of protocol P which is accompanied by the assignment of *true* to agent B's $holds(P, T)$ variable. We then say that agent B obtained tag T, apparently from agent A, in state $\Sigma(t)_{i+1}$. Similarly, $t_i = \mathsf{release}_P(A, T, B)$ denotes any event related to the signal in which agent A releases tag T, apparently to agent B, i.e. assigns *false* to agent A's $holds(P, T)$ variable. We call such an event t_i a release event.

Remark 1. For secure ownership it is important to place the release and obtain signals in the correct position in the ownership transfer protocol. The release signal is placed at a point causally preceding a tag's ownership update, typically at the start of the role for the current owner of the tag. The obtain signal is placed at a point causally following a tag's confirmed ownership update, thus typically at the end of the role for the new owner. It is easy to see that if a release signal appears too late or an obtain signal appears too early, an agent may be holder of a tag while not owning the tag, thus violating secure ownership.

4.2 Secure Ownership Transfer

We say that a set of protocols provides *secure ownership transfer*, if, whenever an agent R becomes owner of a tag, it must be as a result of an execution of an ownership transfer protocol, i.e. the ownership change must be intentional.

To capture an agent's intention to give up ownership, we require that every change in ownership, making R owner of T, must be preceded by a release signal.

We restrict the relation between ownership changes and release signals in two ways. First, the ownership change must be in a one-to-one correspondence with the release signals, i.e. one release signal must not be the source of two or more ownership changes. Second, no corresponding release and ownership-change events related to T may interleave other corresponding release and ownership-change events of T. That is, the one-to-one map must be such that the ownership change for T is mapped to the latest preceding release signal for T.

For tags owned by the intruder, these requirements cannot be enforced. Therefore, an agent R can become owner of a tag, either as a consequence of the tag being intentionally released to R or as a consequence of the tag being released to the agent E controlled by the intruder. In the latter case the intruder must have made R the new owner without properly releasing the tag.

Definition 6 (Secure Ownership Transfer). *Let* Event *denote the set of all possible events and let* $E \in \mathsf{Agent}$ *be the agent controlled by the intruder. A set of protocols* Π *provides secure ownership transfer with respect to* P *if and only if*

$$\forall_{t \in \mathsf{traces}(\Pi)}\ \exists_{f:\mathsf{Event}\to\mathsf{Event},injective}\ \forall_{0\le k<|t|}\ \forall_{R,T\in\mathsf{Agent}}$$
$$\neg\mathsf{owns}_P(R,T,\Sigma(t)_k) \wedge \mathsf{owns}_P(R,T,\Sigma(t)_{k+1}) \Rightarrow$$
$$\exists_{0\le i\le k}\ f(t_k) = t_i \wedge \neg\exists_{i<j\le k}\ t_j = \mathsf{release}_P(*,T,*) \wedge$$
$$(t_i = \mathsf{release}_P(*,T,R) \vee t_i = \mathsf{release}_P(*,T,E)),$$

where $*$ *is used to represent any agent.*

4.3 The Yoon and Yoo Protocol

We demonstrate our definitions on the recently published ownership transfer protocol by Yoon and Yoo [5].

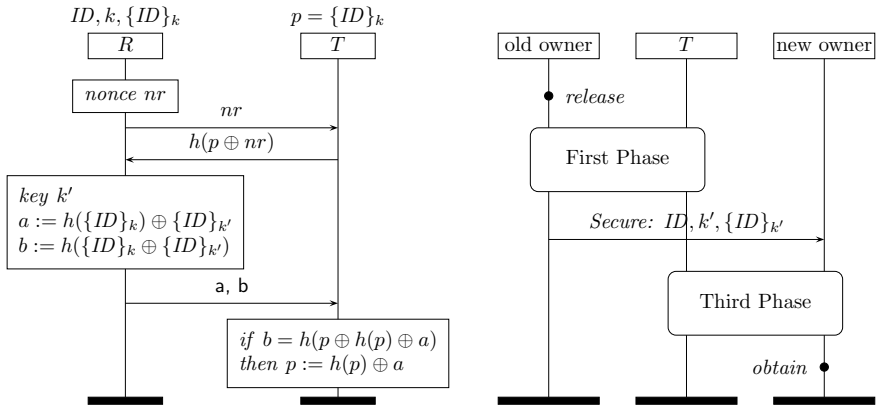

Fig. 1. Flawed ownership transfer protocol [5]

The protocol relies on a shared secret $p = \{ID\}_k$ between owner and tag, called a *pseudonym*. It consists of three phases as shown on the right in Figure 1. The first and the third phase are instantiations of the protocol shown on the left in Figure 1. In the first phase, the old owner updates the pseudonym p, using a fresh key k'. This key together with the real identity and the pseudonym are sent over a secure channel to the new owner in the second phase. The final phase consists of another pseudonym update executed by the new owner and the tag using a fresh key.

Following Remark 1, we put the release signal at the start of the first phase, and the obtain signal at the end of the third phase. Since the pseudonym p of the tag is all that is used in communication with the tag, we take as ownership test

protocol a proof-of-knowledge protocol of p. We can now analyze the protocol with respect to secure ownership and secure ownership transfer.

Consider an execution of the protocol by R, T, and R', where initially R is the owner of the tag T and intends R' to become the new owner. We first show that the protocol does not satisfy secure ownership transfer, because an intruder E can obtain ownership of the tag without being the intended new owner. To achieve this, the intruder queries the target tag T with the constant 0 to which the tag replies with $h(p)$. By eavesdropping on the first phase of the protocol execution, the intruder obtains $a = h(p) \oplus \{ID\}_{k'}$. As soon as the tag updates its pseudonym to $\{ID\}_{k'}$ the intruder becomes owner of the tag.

Next, we show that secure ownership can be violated using knowledge of the tag's pseudonym the intruder has gained after the first phase of the protocol through the previous attack. The intruder eavesdrops on the third phase of the transfer, carried out by T and R'. The new owner R' becomes holder of the tag when the third phase finishes. Using the information learned during this phase the intruder can derive the new pseudonym as he did in the previous attack. The intruder then executes the pseudonym update protocol to update the tag's pseudonym to a pseudonym the new owner R' does not know. Therefore R' loses ownership while still being holder of the tag which violates secure ownership.

Finally, by eavesdropping on the third phase of the ownership transfer, a dishonest previous owner will be able to learn the new pseudonym. Therefore it will not lose ownership and hence exclusive ownership is not satisfied either.

5 Desynchronization

As an application of our definitions we study desynchronization attacks on stateful protocols. Although it is easy to characterize desynchronization for a given protocol (by inspection of the values of the involved variables), it is not straightforward to transform this into a generic definition of desynchronization. In this section we demonstrate how the notion of ownership can be used to define desynchronization.

The execution of a stateful RFID protocol frequently ends with reader and tag updating shared information. An attacker may attempt to disrupt the communication between reader and tag such that the two agents' updates are not correlated. A flawed protocol will not allow the agents to recover from this disruption and the reader and tag will be in a state of *desynchronization*: they will no longer be able to successfully communicate with each other. We call a protocol that is not vulnerable to this type of attack *desynchronization resistant*.

In general, stateful RFID authentication protocols do not need to verify ownership requirements, since the owner of a tag never changes. We argue, however, that our notion of ownership is closely related to desynchronization resistance. Indeed, if there does not exist a reader that can successfully communicate with a tag using a protocol P, then the tag has no owners with respect to P.

We say that a protocol P is desynchronization resistant, if a tag never loses all its owners with respect to P.

Definition 7 (Desynchronization Resistance). *A protocol $P \in \Pi$ is desynchronization resistant if and only if*

$$\forall_{t \in \mathsf{traces}(\Pi)}\ \forall_{0 \leq i < |t|}\ \forall_{T \in \mathsf{Agent}} \\ \exists_{R \in \mathsf{Agent}}\ \mathsf{owns}_P(R, T, \Sigma(t)_i) \Rightarrow \exists_{R' \in \mathsf{Agent}}\ \mathsf{owns}_P(R', T, \Sigma(t)_{i+1}).$$

It is interesting to note that desynchronization resistance together with exclusive ownership can imply secure ownership. Therefore in order to prove secure ownership with respect to a test protocol P it is sufficient, under the conditions stated in the following theorem, to prove desynchronization resistance of P and exclusive ownership with respect to P. Note that the second condition in the theorem corresponds to placing obtain signals in protocols at a point in which an agent is sure to have become owner of a tag, as described in Remark 1.

Theorem 1. *Let Π be a set of protocols containing the test protocol P. Suppose that Π provides exclusive ownership with respect to P and that P is desynchronization resistant. Then Π provides secure ownership for every trace which satisfies the following two conditions.*

(1) *In the initial state every holder of a tag is owner of the tag.*
(2) *An agent only becomes holder of a tag if it owns the tag.*

Proof. Suppose towards a contradiction that there is a trace $t \in \mathsf{traces}(\Pi)$ such that in a state $\Sigma(t)_i$ an agent R holds a tag T, but does not own the tag. By condition (2) the agent has not become holder of T in state $\Sigma(t)_i$. Thus there must be a state $\Sigma(t)_j$, $1 \leq j < i$, in which the agent became holder of the tag. By exclusive ownership, no other agent owns the tag in state $\Sigma(t)_i$. Desynchronization resistance implies that if no agent owns T in a state $\Sigma(t)_i$, then no agent could have owned T in state $\Sigma(t)_{i-1}$. By condition (2) no agent could have become holder in state $\Sigma(t)_{i-1}$. This argument can be repeated to conclude that no agent could have owned T in the initial state and no agent could become holder in the states $\Sigma(t)_1, \ldots, \Sigma(t)_i$. Thus R must have been the holder in the initial state. This contradicts condition (1).

5.1 The Song and Mitchell Protocol

Song and Mitchell [6] propose a stateful RFID protocol that relies on a shared secret for authentication. Their protocol achieves identification and authentication of the tag and can therefore be used in scenarios like supply chain management or access control. They notice that in many proposed protocols tags and readers can be desynchronized by blocking certain messages from reader to tag. They attempt to prevent desynchronization attacks by storing additional information, allowing the reader to re-synchronize with a tag in case messages are blocked. In this section we show that this mechanism is insufficient to provide desynchronization resistance by describing an attack that has previously gone unnoticed.

We demonstrate that by modifying and blocking certain messages an attacker can force a tag and reader to carry out differing updates of their shared secret. As a result, the reader loses ownership of the tag.

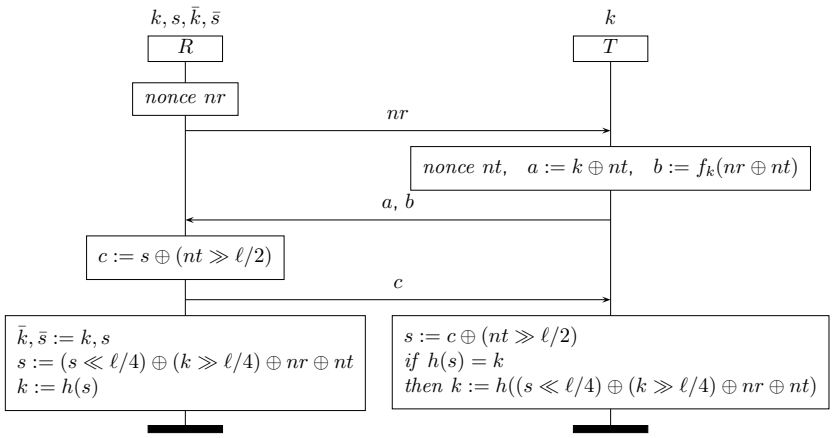

Fig. 2. RFID authentication protocol for low-cost tags [6]

The protocol specification is given in Figure 2. We use $f_k(\cdot)$ to denote a keyed hash function, $a \gg b$, $a \ll b$ to denote a cyclic right and left shift, respectively, of a over b bits, and ℓ to denote the bit length of the value to be shifted.

We assume that the attacker does not know the shared secret between tag and reader. The attacker eavesdrops on the first two messages (nr and a, b) and then aborts the protocol by blocking the third message (c). The tag has not successfully completed its run and therefore does not carry out its update. The attacker then challenges the same tag with his own nonce ni. The tag responds with a', b', where $a' = k \oplus nt'$ and $b' = f_k(ni \oplus nt')$. Using distributivity of $\oplus$ over $\gg$, the attacker can now construct a valid reader response $c' = c \oplus ((a \oplus a') \gg \ell/2) = s \oplus (nt' \gg \ell/2)$. The tag accepts the message and updates its k to $h((s \ll \ell/4) \oplus (k \gg \ell/4) \oplus ni \oplus nt')$. As soon as the tag carries out its update the genuine reader loses ownership. Indeed, no agent can successfully complete the test protocol, since the key k is unknown (even to the attacker). Thus, the protocol is not desynchronization resistant.

6 Related Work

Work on ownership transfer in RFID systems has thus far mostly focused on designing ownership transfer protocols, but not on their security requirements. A notable exception is the work by Song [7]. It provides a first survey of security requirements related to ownership transfer. Song also proposes a set of protocols for secure ownership transfer based on earlier work by Song and Mitchell [6]. However, this set of protocols suffers from the same flaws that are described in Section 5 and by Van Deursen and Radomirović [8].

The first treatment of ownership transfer in RFID systems is due to Molnar et al. [9]. They describe a protocol that relies on a trusted center. Readers send tag pseudonyms to the center requesting the real identity of a tag. If the reader

is the owner of the tag it receives the identity. Owners of tags can ask the trusted center to transfer the ownership of a tag to a new owner. The trusted center subsequently refuses identity requests from the old owner, and accepts them from the new owner. A trusted party is also used by the protocol of Saito et al. [10]. Here, the trusted party shares a key with the tag which is used to update the owner's key. Hence an ownership transfer consists of a request to the trusted party to encrypt the new owner's key for the tag.

Osaka et al. [11] are among the first to propose a two-party ownership transfer protocol. Lei and Cao [12], Jäppinen and Hämäläinen [13], and Yoon and Yoo [5] describe a flaw in the protocol by Osaka et al. and propose an improved version of the protocol. We describe an attack on Yoon and Yoo's protocol in Section 4.2.

Lim and Kwon [14] propose a protocol which, compared to other solutions, uses a more computationally intensive mutual authentication method based on key chains. Solutions based on symmetric encryption have also been proposed by Fouladgar and Afifi [15] and Koralalage et al [16]. Finally, one of the most recent protocols in this area is due to Dimitriou [17]. Its distinguishing feature is that it enables the owner of a tag to revert the tag to its original state. This is useful for after-sales services, since it makes it possible for the tag's new owner to let a retailer recognize a sold tag.

7 Conclusion and Future Work

We have presented formal definitions of ownership and ownership transfer, as well as their secure variants. We have demonstrated the applicability of our definitions by exhibiting attacks on secure ownership, exclusive ownership, and secure ownership transfer on a recently proposed ownership transfer protocol [5]. As an application of our definitions we have formalized desynchronization resistance. We have used this formalization to uncover a flaw in a stateful RFID protocol [6].

While we consider a formal definition of ownership to be of independent interest, it will clearly become much more valuable when combined with existing security and privacy properties. For instance, in a parcel delivery system, where RFID tags are attached to parcels, *non-repudiation* for obtaining ownership of RFID tags and *untraceability* of these tags by unauthorized entities become important. We have only briefly indicated the connections between untraceability and exclusive ownership. A useful next step is to study conditions under which untraceable protocols can be safely composed with ownership transfer protocols. This requires in particular an investigation into the interplay between two or more untraceable protocols out of a set of protocols.

Another direction concerns the construction of ownership transfer protocols and proofs of their correctness. The model used in this work has been designed in such a way that the verification of our security requirements should be possible with a model checking tool.

Acknowledgments. We are grateful to Carst Tankink, Erik de Vink, and the anonymous reviewers for their valuable comments which helped to improve this work.

References

1. Cremers, C., Mauw, S.: Operational semantics of security protocols. In: Scenarios: Models, Algorithms and Tools (Dagstuhl 03371 post-seminar proceedings, September 7–12, 2003). Volume 3466 of Lecture Notes in Computer Science. (2005) 66–89
2. Ryan, P., Schneider, S., Goldsmith, M., Lowe, G., Roscoe, B.: Modelling and Analysis of Security Protocols. Addison-Wesley Professional (2001)
3. Dolev, D., Yao, A.: On the security of public key protocols. IEEE Transactions on Information Theory **IT-29**(2) (1983) 198–208
4. Rudolph, E., Graubmann, P., Grabowski, J.: Tutorial on message sequence charts. Computer Networks and ISDN Systems **28**(12) (1996) 1629–1641
5. Yoon, E., Yoo, K.: Two security problems of RFID security method with ownership transfer. In: Proc. IFIP International Conference on Network and Parallel Computing, IEEE Computer Society (2008) 68–73
6. Song, B., Mitchell, C.: RFID authentication protocol for low-cost tags. In: Proc. First ACM Conference on Wireless Network Security, ACM (2008) 140–147
7. Song, B.: RFID tag ownership transfer. In: Proc. Workshop on RFID Security. (2008)
8. van Deursen, T., Radomirović, S.: Attacks on RFID protocols. Cryptology ePrint Archive, Report 2008/310 (2008) http://eprint.iacr.org/.
9. Molnar, D., Soppera, A., Wagner, D.: A scalable, delegatable pseudonym protocol enabling ownership transfer of RFID tags. In: Proc. Selected Areas in Cryptography. Volume 3897 of Lecture Notes in Computer Science., Springer (2005) 276–290
10. Saito, J., Imamoto, K., Sakurai, K.: Reassignment scheme of an RFID tag's key for owner transfer. In: Proc. Embedded and Ubiquitous Computing. Volume 3823 of Lecture Notes in Computer Science., Springer (2005) 1303–1312
11. Osaka, K., Takagi, T., Yamazaki, K., Takahashi, O.: An efficient and secure RFID security method with ownership transfer. In: Proc. Computational Intelligence and Security, Springer-Verlag (2006) 778–787
12. Lei, H., Cao, T.: RFID protocol enabling ownership transfer to protect against traceability and dos attacks. In: Proc. The First International Symposium on Data, Privacy, and E-Commerce, IEEE Computer Society (2007) 508–510
13. Jäppinen, P., Hämäläinen, H.: Enhanced RFID security method with ownership transfer. In: Proc. International Conference on Computational Intelligence and Security, IEEE Computer Society (2008) 382–385
14. Lim, C., Kwon, T.: Strong and robust RFID authentication enabling perfect ownership transfer. In: Proc. Conference on Information and Communications Security. Volume 4307 of Lecture Notes in Computer Science., Springer (2006)
15. Fouladgar, S., Afifi, H.: A simple privacy protecting scheme enabling delegation and ownership transfer for RFID tags. Journal of Communications **2** (2007) 6–13
16. Koralalage, K., Reza, S.M., Miura, J., Goto, Y., Cheng, J.: POP method: an approach to enhance the security and privacy of RFID systems used in product lifecycle with an anonymous ownership transferring mechanism. In: Proc. ACM Symposium on Applied Computing, ACM (2007) 270–275

17. Dimitriou, T.: rfidDOT: RFID delegation and ownership transfer made simple. In: Proc. 4th International Conference on Security and Privacy in Communication Networks, ACM (2008) 1–8
18. Fokkink, W.: Introduction to Process Algebra. Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag (2000)

A Syntax and Semantics of RFID Protocols

A.1 Protocol Specifications

A *protocol* is a map from an n-tuple of distinct *roles* to an n-tuple of *role specifications*. A role specification consists of a declaration of the nonces and variables (defined below) used by that role and the *events* defining the messages that an honest agent sends and expects to read, when executing the role. Events can be composed in three ways. *Sequential composition*, denoted by $(\_ \cdot \_)$, specifies consecutive execution of events while *alternative composition*, denoted by $(\_ + \_)$, models branching. *Conditional branching*, denoted by $(\_ \lhd x = y \rhd \_)$, chooses the left branch if $x = y$ and the right branch otherwise.

Messages to be sent over the network are constructed by a term algebra. We define *Agent* to be the set of agent names allowed to execute protocols. The set of constants, *Const*, contains values that are globally known, such as the natural numbers. The set *Nonce* contains nonces, i.e. values that are freshly generated for every protocol execution. Functions are contained in the set $\mathcal{F}$.

We consider four pairwise disjoint sets of variables. The set *RoleName* contains the role names of the roles in the protocol. During protocol execution, role names are instantiated by the names of the agents executing the protocol. Local variables are variables that are instantiated during an execution of a run, but lose their value after the run finishes. They are contained in Var_L. The set Var_G contains global variables which represent the persistent knowledge of an agent. Their values are maintained across protocol runs. Global variable arrays, contained in $\mathcal{G}$, are a generalization of global variables. They group global variables, such as agent's public keys, in order to simplify role specifications. We use a special variable θ to denote the identifier of a run. This variable is used to disambiguate nonces from different runs. A fresh value is assigned to θ when a role is instantiated. Note that θ must not occur in any of the variable sets.

Complex terms can be constructed by pairing terms, denoted by $(\_, \_)$, encrypting a term by another term, denoted by $\{\_\}_\_$, or applying a function $f \in \mathcal{F}$ to a term, denoted by $f(\_)$.

Send and read events can be accompanied by a list of variable *assignments*. Assignments can be done to global variables and to global variable arrays. Execution of a send or read event accompanied by assignment of variables is considered to be an atomic step.

Inspired by Ryan et al. [2], we use *signals* to indicate that a certain point in the protocol has been reached.

A.2 Protocol Execution

In this section we describe how, through instantiation of variables, an abstract role specification can be transformed into an execution by an agent. Furthermore, we define how the interleaved execution of a collection of runs defines the behavior of a system.

A system state $\langle A, G, SB, RB, I\rangle$ is determined by the active runs A, the global knowledge of the agents G, the send buffer SB, the read buffer RB, and the intruder's knowledge I. An active run contains a *run identifier*, the name of the *agent* executing the run, a list of remaining *events*, as well as the local variable assignment for that run. The *global knowledge* contains the global variable assignment for every agent. Since we assume communication between agents to be asynchronous, agents write messages to a *send buffer* and read messages from a *read buffer*. The *intruder knowledge* contains the set of terms that the intruder initially knows, extended with the terms learned during protocol executions.

The behavior of the system is defined as a transition relation between system states. The derivation rules, depicted in Figures 3, 4, and 5, are of the form

$$\frac{C}{S \xrightarrow{l} S'},$$

expressing that a system in state S can do a transition to state S' with label l if condition C is satisfied. A state transition is the conclusion of applying one of these rules. In this way, starting from an initial state $\langle \emptyset, \emptyset, \emptyset, \emptyset, M_0\rangle$, where M_0 denotes the initial intruder knowledge, we can derive all possible behavior of a system executing a set of protocols.

We separate the derivation rules into three categories. The agent rules (Figure 3) express under which conditions an agent may execute one of its protocol steps. Agent rules can be composed in several ways to model possible protocol flow, expressed by the composition rules (Figure 4). Finally, the intruder rules (Figure 5) model the capabilities of the intruder.

Agent Rules. The *create*-rule creates a run with a fresh run identifier f and adds it to the set of active runs. We use $\mathit{runids}(A)$ to denote the set of run identifiers in A. We capture the set of agents that is allowed to execute role R by $\mathit{agentsof}(R)$. This is to optimize the verification of protocols in which agents only implement a subset of the protocol roles. The *type* of an agent refers to the possibility of the agent to be active in at most one run ($\mathit{type} = 1$) or more than one run at a time ($\mathit{type} = *$). We denote the set of agents that currently have an unfinished run by $\mathit{unfinished}(A)$. The new active run is a tuple containing the run identifier f, the agent name n, the events of the role (denoted by $\mathit{eventsof}(R)$) and the initial local variable assignment. The variable assignment maps the role name to the agent name ($R \mapsto n$) and the run identifier variable to its fresh value ($\theta \mapsto f$).

The execution state of a run can be determined by inspecting its list of events. An agent has successfully completed a run when this list is empty (denoted

$$[\mathsf{create}]\ \frac{\begin{array}{c} n \in agentsof(R) \quad ((n \notin unfinished(A) \wedge type(n) = 1) \vee type(n) = *) \\ f \notin runids(A) \quad a = (f, n, eventsof(R), \{R \mapsto n, \theta \mapsto f\}) \end{array}}{\langle A, G, SB, RB, I\rangle \xrightarrow{create(f,R,n)} \langle A \cup \{a\}, G, SB, RB, I\rangle}$$

$$[\mathsf{send}]\ \frac{\begin{array}{c} x \xrightarrow{send(m)[\overrightarrow{x}:=\overrightarrow{c}]/T/F/} x' \quad a = (f, n, x, \rho) \in A \quad a' = (f, n, x', \rho) \\ G(n) = \sigma \quad \sigma' = \sigma[\overrightarrow{c}/\overrightarrow{x}] \quad \forall_{(v,w)\in T}\ \sigma\rho(v) = \sigma\rho(w) \quad \forall_{(v,w)\in F}\ \sigma\rho(v) \neq \sigma\rho(w) \end{array}}{\langle A, G, SB, RB, I\rangle \xrightarrow{send(f,\sigma\rho(m))} \langle A\backslash\{a\} \cup \{a'\}, G[\sigma'/n], SB \cup \{\sigma\rho(m)\}, RB, I\rangle}$$

$$[\mathsf{read}]\ \frac{\begin{array}{c} x \xrightarrow{read(m)[\overrightarrow{x}:=\overrightarrow{c}]/T/F/} x' \quad a = (f, n, x, \rho) \in A \\ Match_{\rho,\sigma}(m, m', \rho') \quad m' \in RB \\ G(n) = \sigma \quad \sigma' = \sigma[\overrightarrow{c}/\overrightarrow{x}] \quad \forall_{(v,w)\in T}\ \sigma\rho(v) = \sigma\rho(w) \quad \forall_{(v,w)\in F}\ \sigma\rho(v) \neq \sigma\rho(w) \end{array}}{\langle A, G, SB, RB, I\rangle \xrightarrow{read(f,m')} \langle A\backslash\{a\} \cup \{(f, n, x', \rho)\}, G[\sigma'/n], SB, RB\backslash\{m'\}, I\rangle}$$

$$[\mathsf{end}]\ \frac{a = (f, n, x, \rho) \in A \quad x \neq \epsilon}{\langle A, G, SB, RB, I\rangle \xrightarrow{end(f)} \langle A\backslash\{a\} \cup \{(f, n, \bot \cdot x, \rho)\}, G, SB, RB, I\rangle}$$

Fig. 3. Agent rules

by ϵ). An event list which has been marked (with $\bot$), by means of the *end*-rule, indicates that the run has been terminated before it was able to finish successfully. Otherwise the run is currently unfinished.

Any agent executing a *send* event, thereby changing from state x to x' (for x and x' lists of events), changes the overall system state. The sent message (obtained by applying the local variable assignment ρ and global variable assignment σ to the message) is added to the send buffer.

A send event can be accompanied by a list of global variable assignments of the form $x := c$. We denote by $\overrightarrow{x} := \overrightarrow{c}$ the simultaneous assignment of a list of variables x to a list of values c of the same length. The rule changes the current global variable assignment σ to $\sigma[\overrightarrow{c}/\overrightarrow{x}]$, where $\sigma[c/x]$ denotes the substitution σ altered such that $x \mapsto c$. When the execution of the send event is part of a (nested) conditional branching statement, a (number of) equalities (T) and/or inequalities (F) have to be fulfilled. Each of these (in)equalities must hold after replacing the local and global variables with their respective values.

An agent executing a *read* event changes the system state similar to a send event. It takes a message m' from the read buffer and matches it against the message that an agent expects to receive. It furthermore extends the local variable assignment ρ to ρ' such that any free variables in the expected message are assigned a value making $\sigma(m)$ and m' equivalent. Finally, the message m' is removed from the read buffer.

The purpose of the match predicate, used in the read-rule, is to fix a minimal substitution ρ' that maps every variable in m to a ground term, such that $\sigma\rho(m) = m'$. Furthermore, the term m' is required to be readable. Formally,

$$\begin{aligned} Match_{\rho,\sigma}(m, m', \rho') \equiv\ & m' = \sigma\rho(m) \wedge dom(\rho) = vars(m) \wedge \\ & Rd(rng(\rho) \cup rng(\sigma), \rho', \sigma(m), m'). \end{aligned}$$

The readability predicate Rd decides whether a given term is readable. A received term m' is readable with respect to an expected term m if there is a substitution ρ that makes them syntactically equivalent. Furthermore, every subterm required to read the term must be inferable from the agent's knowledge extended with the received message. More formally, let $m, p \in Term$, $K \in \mathcal{P}(Term)$, and $\rho(m) = m'$, then

$$Rd(K, \rho', m, m') \equiv \forall_{a \sqsubseteq m}\ K \cup \{m'\} \vdash \rho(a) \vee K \cup \{m'\} \vdash \rho(a)^{-1}.$$

The subterm operator, denoted by $\sqsubseteq$, is used to decompose a term into the terms from which it was constructed. Let $t, t_1, t_2 \in Term$, then:

$$\begin{array}{lll} t \sqsubseteq t & t_1 \sqsubseteq (t_1, t_2) & t_2 \sqsubseteq (t_1, t_2) \\ t_1 \sqsubseteq \{t_1\}_{t_2} & t_2 \sqsubseteq \{t_1\}_{t_2} & t \sqsubseteq h(t) \end{array}$$

Composition Rules. The rules in Figure 4 describe the semantics for composition of events. They are very similar to the transition rules for Basic Process Algebra [18]. The main difference is the treatment of the conditional branching statement $x \lhd v = w \rhd y$. Instead of requiring $v = w$ (or $v \neq w$) as a premise we add it as a proof obligation. We therefore have rules of the form

$$\frac{A}{x \xrightarrow{a/T/F} x'},$$

stating that an agent in state x can execute a and transition to x', if the premise A is satisfied. The execution of a additionally introduces the proof obligations in T (equalities) and F (inequalities).

In the following, let a be a read, send, or claim event and x and y be variables ranging over lists of events. The *exec* rule states that an event a can be successfully executed introducing no proof obligations. The *choice* rules express that in an alternative composition either of the branches can be executed. The sequential composition states that when executing $x \cdot y$, first x is executed and then y. The conditional branching statement $x \lhd v = w \rhd y$ expresses that the left branch can be executed, introducing a proof obligation $v = w$, or the right branch can be executed, introducing a proof obligation $v \neq w$.

Intruder Rules. The rules in Figure 5 describe the capabilities of the intruder. The intruder operates on the send and read buffer (SB and RB). The *deliver* rule transfers a message from the send buffer to the read buffer. If the intruder has eavesdropping capabilities he may additionally add that message to his knowledge, as stated by the *eavesdrop* rule. The *block* rule expresses that any message in the send buffer may be removed by the intruder, but the intruder still learns the message. The intruder may also be able to *inject* messages, that is, add messages he can infer from his knowledge to the read buffer.

Different adversaries can be modeled by selecting a subset of the rules in Figure 5. An adversary with no powers is modeled by having only the deliver

$$[\mathsf{exec}]\ \frac{}{a \xrightarrow{a/\emptyset/\emptyset} \checkmark}$$

$$[\mathsf{choice}_1]\ \frac{x \xrightarrow{a/T/F} x'}{x + y \xrightarrow{a/T/F} x'} \qquad [\mathsf{choice}_2]\ \frac{y \xrightarrow{a/T/F} y'}{x + y \xrightarrow{a/T/F} y'}$$

$$[\mathsf{seq}_1]\ \frac{x \xrightarrow{a/T/F} x'}{x \cdot y \xrightarrow{a/T/F} x' \cdot y} \qquad [\mathsf{seq}_2]\ \frac{x \to \checkmark \quad y \xrightarrow{a/T/F} y'}{x \cdot y \xrightarrow{a/T/F} y'}$$

$$[\mathsf{cond}_1]\ \frac{x \xrightarrow{a/T/F} x'}{x \lhd v = w \rhd y \xrightarrow{a/T \cup (v,w)/F} x'} \qquad [\mathsf{cond}_2]\ \frac{y \xrightarrow{a/T/F} y'}{x \lhd v = w \rhd y \xrightarrow{a/T/F \cup (v,w)} y'}$$

Fig. 4. Composition rules

rule. A passive adversary can be modeled by additionally having the eavesdrop rule. The Dolev-Yao intruder [3], which is an adversary that essentially controls the network, is modeled by the union of the four rules.

$$[\mathsf{deliver}]\ \frac{m \in S}{\langle A, G, S, R, I\rangle \xrightarrow{deliver} \langle A, G, S \backslash \{m\}, R \cup \{m\}, I\rangle}$$

$$[\mathsf{block}]\ \frac{m \in S}{\langle A, G, S, R, I\rangle \xrightarrow{block} \langle A, G, S \backslash \{m\}, R, I \cup \{m\}\rangle}$$

$$[\mathsf{inject}]\ \frac{I \vdash m}{\langle A, G, S, R, I\rangle \xrightarrow{inject} \langle A, G, S, R \cup \{m\}, I\rangle}$$

$$[\mathsf{eavesdrop}]\ \frac{m \in S}{\langle A, G, S, R, I\rangle \xrightarrow{eavesdrop} \langle A, G, S \backslash \{m\}, R \cup \{m\}, I \cup \{m\})}$$

Fig. 5. Intruder rules